Designing a Highly Expressive Algorithmic Music Composition System for Non-Programmers

DMRN+11: DIGITAL MUSIC RESEARCH NETWORK
ONE-DAY WORKSHOP 2016

QUEEN MARY UNIVERSITY OF LONDON
TUE 20 DECEMBER 2016

Designing a Highly Expressive Algorithmic Music Composition System for Non-Programmers

Matt Bellingham[1*], Simon Holland[2] and Paul Mulholland[3]

[1*]Department of Music, University of Wolverhampton, UK, matt.bellingham@wlv.ac.uk
[2]Music Computing Lab, Centre for Research in Computing, The Open University, UK
[3]Knowledge Media Institute, Centre for Research in Computing, The Open University, UK

***Abstract*—Algorithmic composition systems allow for the partial or total automation of music composition by formal, computational means. Typical algorithmic composition systems generate nondeterministic music, meaning that multiple musical outcomes can result from the same algorithm - consequently the output is generally different each time the algorithm runs.**

Here we present an algorithmic composition system designed to meet the needs of a particular user group: undergraduate Music Technology students. Unexpectedly, the specific needs of this user group led us to radical design decisions, which ended up reshaping the fundamentals of the underlying programming language design. Our users are typically not programmers, and they are often not traditional musicians. While they may be conversant with some elements of music theory, their background is often as self-taught music producers with experience of making music electronically using music sequencers/DAWs.

There are many existing tools which can be used for algorithmic music composition, but from the perspective of our target user group, they all exhibit various limitations [1, 2]. For example, Ableton Live offers a simple and efficient UI which provides a number of tools to assist loop-based composition, but the software lacks the expressivity and generality required for algorithmic work. Graphical programming languages such as Max and Pure Data are highly expressive but require the user to have sufficient pre-existing musical knowledge to build their own musical structure into patches. Text-oriented languages such as SuperCollider and Sonic Pi offer high expressivity and are well adapted to musical structures, but require the user to manipulate musical materials through structures and syntax shaped primarily by the concerns of conventional programming languages.

Our system is designed to meet several needs: to enable our target users to create algorithmic music of arbitrary complexity; to facilitate graphical programming with minimal syntactical concerns; and to make common musical tasks simple. As a by-product, these properties promote learning the concepts of algorithmic composition via hands-on experience.

In the new algorithmic composition system, the principal programming primitive is called a *chooser*. Every instance of this primitive affords such musically useful actions as: loop X until Y finishes; hierarchical organization; random selection; and choice points. Due to this and related design decisions, the system has relatively low viscosity and low verbosity: i.e. musically desirable changes are relatively easy to make, and musically complex constructs can be expressed concisely. These properties derive from what is technically known as the 'closeness of mapping' of the notation, i.e. how closely the notation corresponds to the problem world [3]. In part, this is achieved because the programming language gives the affordances of the graphical track/mixer view of a sequencer/DAW while providing the full power of a recursive programming language.

At present the design is being iteratively refined using the programming walkthrough method [4] and implemented using SuperCollider as the back end.

REFERENCES

[1] Bellingham, M., Holland, S. and Mulholland, P. (2014a) A Cognitive Dimensions analysis of interaction design for algorithmic composition software, In *Proceedings of Psychology of Programming Interest Group Annual Conference 2014*, du Boulay, B. and Good, J. (eds.), University of Sussex, pp. 135–140, [online] Available from: http://www.sussex.ac.uk/Users/bend/ppig2014/15ppig2014_submission_10.pdf.

[2] Bellingham, M., Holland, S. and Mulholland, P. (2014b) *An analysis of algorithmic composition interaction design with reference to cognitive dimensions*, The Open University, [online] Available from: http://computing-reports.open.ac.uk/2014/TR2014-04.pdf.

[3] Green, T. R. and Petre, M. (1996) Usability Analysis of Visual Programming Environments: a `cognitive dimensions' framework, *Journal of Visual Languages and Computing*, 7, pp. 131–174.

[4] Bell, B., Citrin, W. V., Lewis, C. and Rieman, J. (1992) The Programming Walkthrough: A Structured Method for Assessing the Writability of Programming Languages; CU-CS-577-92, *Computer Science Technical Reports*, Paper 554, [online] Available from: http://scholar.colorado.edu/csci_techreports/554.